Information and Computation 209 (2011) 486–512

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Rigid tree automata and applications[☆]

Florent Jacquemard[a,*], Francis Klay[b], Camille Vacher[c]

[a] INRIA Saclay - Île-de-France, LSV, ENS Cachan, 61 avenue du Président Wilson, 94235 Cachan Cedex, France
[b] Orange DPS/2IA/RSF, FA140, R&D Lannion 2, avenue Pierre Marzin, 22307 Lannion Cedex, France
[c] LIFL, Univ. Lille I, INRIA Lille - Nord Europe, Parc Scientifique de la Haute Borne, Park Plazza Bât A - 40 avenue Halley, 59650 Villeneuve d'Ascq, France

ARTICLE INFO

Article history:
Available online 18 November 2010

Keywords:
Tree automata
Symbol constraints
Term rewriting
Verification

ABSTRACT

We introduce the class of rigid tree automata (RTA), an extension of standard bottom-up automata on ranked trees with distinguished states called rigid. Rigid states define a restriction on the computation of RTA on trees: RTA can test for equality in subtrees reaching the same rigid state. RTA are able to perform local and global tests of equality between subtrees, non-linear tree pattern matching, and some inequality and disequality tests as well. Properties like determinism, pumping lemma, Boolean closure, and several decision problems are studied in detail. In particular, the emptiness problem is shown decidable in linear time for RTA whereas membership of a given tree to the language of a given RTA is NP-complete. Our main result is the decidability of whether a given tree belongs to the rewrite closure of an RTA language under a restricted family of term rewriting systems, whereas this closure is not an RTA language. This result, one of the first on rewrite closure of languages of tree automata with constraints, is enabling the extension of model checking procedures based on finite tree automata techniques, in particular for the verification of communicating processes with several local non-rewritable memories, like security protocols. Finally, a comparison of RTA with several classes of tree automata with local and global equality tests, with dag automata and Horn clause formalisms is also provided.

1. Introduction

Tree automata (TA) are finite representations of infinite sets of terms. In automated theorem proving, they allow to cut infinite computation branches by reduction to TA decision problems. In system and software verification, TA can be used to represent infinite sets of states of a system or a program (in the latter case, a term can represent the program itself), messages exchanged in a communication protocol, XML documents, etc. In these settings, the closure properties of TA languages permit incremental constructions and verification problems can be reduced to TA problems decidable in polynomial time like emptiness (is the language recognized by a given TA empty) and membership (is a given term t recognized by a given TA).

Despite these nice properties, a big limitation of TA is their inability to test equalities between subterms during their computation: TA are able to detect linear patterns like $\mathsf{fst}(\mathsf{pair}(x_1, x_2))$ but not a pattern like $\mathsf{pair}(x, x)$. Several extensions of TA have been proposed to overcome this problem, by addition of equality and disequality tests in TA transition rules (the classes [8,16] have a decidable emptiness problem), or an auxiliary memory containing a tree and memory comparison [12].

☆ This work has been partly supported by the FET-Open Grant Agreement FOX No. FP7-ICT-23359, the INRIA ARC 2010 project ACCESS, and the ANR Sesur 07 project AVOTÉ.

* Corresponding author at: INRIA Saclay – Île de France, France.
E-mail addresses: florent.jacquemard@inria.fr (F. Jacquemard), francis.klay@orange-ftgroup.com (F. Klay), vacher@lsv.ens-cachan.fr (C. Vacher).

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.11.015

Pushdown tree automata [10,23] also permit such tests. However, they are all limited to local tests, at a bounded distance from the current position.

In this paper, we define the *rigid tree automata* (RTA) by the distinction of some states as *rigid*, and the condition that the subterms recognized in one rigid state during a computation are all equal. With such a formalism, it is possible to check local and global equality tests between subterms, and also the subterm relation or restricted disequalities. In Sections 3–7 we study issues like pattern matching, pumping lemmas, compare expressiveness with related classes of automata, determinism, closure of recognized languages under Boolean operations, and decision problems for RTA. RTA are a particular case of the more general class Tree Automata with General Equality and Disequality constraints (TAGED [19], see Section 4.1). The study of the class RTA alone is motivated by the complexity results and specific applications to verification mentioned below. But our most original contribution is the study of the rewrite closure of RTA languages in Section 8.

Term rewriting systems (TRS) is a general formalism for the symbolic evaluation of terms by replacement of some patterns by others, following rewrite rules. Combining tree automata and term rewriting techniques has been very successful in verification, see e.g. [9,21]. In this context, term rewriting systems (TRS) can describe the transitions of a system, the evaluation of a program [9], the specification of operators used to build protocol messages [1] or also transformation of documents. If a TA $\mathcal{A}$ is used to finitely represent an infinite set $L(\mathcal{A})$ of states of a system, the rewrite closure $\mathcal{R}^*\big(L(\mathcal{A})\big)$ of the language $L(\mathcal{A})$ using $\mathcal{R}$ represents the set of states reachable from states described by $\mathcal{A}$. When $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is again a TA language, the verification of a safety property amounts to checking for the existence of an error state in $\mathcal{R}^*\big(L(\mathcal{A})\big)$ (either a given term t or a term in a given regular language). This technique, sometimes referred as regular tree model checking, has driven a lot of attention to the rewrite closure of tree automata languages. However, there has been very few studies of this issue for constrained TA (see e.g. [24]). The reason is the difficulty to capture the behavior of constraints after the application of rewrite rules.

In Section 8, we show that it is decidable whether a given term t belongs to the rewrite closure of a given RTA language for a restricted class of linear TRS called invisibly pushdown, whereas this closure is generally not an RTA language. Linear and invisibly pushdown TRS can typically specify cryptographic operators like $\mathsf{decrypt}(\mathsf{crypt}(x, \mathsf{pk}(A)), \mathsf{sk}(A)) \to x$.

Using RTA instead of TA in a regular tree model checking procedure permits to handle processes with local and global memories taking their values in infinite domains and which can be written only once. We illustrate this idea in Section 9 with the description of a potential application of RTA to the verification of security protocols.

2. Preliminaries

A *signature* Σ is a finite set of function symbols with arity. We write Σ_m for the subset of function symbols of Σ of arity m. Given an infinite set $\mathcal{X}$ of variables, the set of terms built over Σ and $\mathcal{X}$ is denoted $\mathcal{T}(\Sigma, \mathcal{X})$, and the subset of ground terms (terms without variables) is denoted $\mathcal{T}(\Sigma)$. The set of variables occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $vars(t)$ occurs at most once in t. A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ into $\mathcal{T}(\Sigma, \mathcal{X})$. The application of a substitution σ to a term t is the homomorphic extension of σ to $\mathcal{T}(\Sigma, \mathcal{X})$.

A term t can be seen as a function from its set of *positions* $\mathcal{P}os(t)$ into function symbols or variables of $\Sigma \cup \mathcal{X}$. The positions of $\mathcal{P}os(t)$ are sequences of positive integers (ε, the empty sequence, is the root position). Positions are compared wrt the prefix ordering: $p_1 < p_2$ iff there exists $p \neq \varepsilon$ such that $p_2 = p_1 \cdot p$ (where $p_1 \cdot p$ denotes the concatenation of p_1 and p). In this case, p is denoted $p_2 - p_1$. The subterm of t at position p is denoted $t|_p$, and the replacement in t of the subterm at position p by u is denoted $t[u]_p$. The *depth* $d(t)$ of t is the length of its longest position. A *n-context* is a linear term of $\mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$. The application of a *n-context* C to n terms $t_1, \ldots, t_n$, denoted by $C[t_1, \ldots, t_n]$, is defined as the application to C of the substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

2.1. Term rewriting

A *term rewrite system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$ (it is called the left-hand side (*lhs*) of the rule) and $r \in \mathcal{T}(\Sigma, vars(\ell))$ (it is called right-hand side (*rhs*)). A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to $s \in \mathcal{T}(\Sigma, \mathcal{X})$ by a TRS $\mathcal{R}$ (denoted $t \xrightarrow[\mathcal{R}]{} s$) if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position $p \in \mathcal{P}os(t)$ and a substitution σ such that $t|_p = \sigma(\ell)$ and $s = t[\sigma(r)]_p$. In this case, t is called *reducible*. An irreducible term is also called an *$\mathcal{R}$-normal-form*. The transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$. Given $L \subseteq \mathcal{T}(\Sigma, \mathcal{X})$, we denote $R^*(L) = \{t \mid \exists s \in L, s \xrightarrow[R]{*} t\}$. A TRS $\mathcal{R}$ is called *linear* if all the terms in its rules are linear and *collapsing* if every rhs of rules of $\mathcal{R}$ is a variable.

2.2. Tree automata

Following definitions and notations of [13], we consider tree automata which compute bottom-up (from leaves to root) on (finite) ground terms in $\mathcal{T}(\Sigma)$. At each stage of computation on a tree t, a tree automaton reads the function symbol f at the current position p in t and updates its current state, according to f and the respective states reached at the positions immediately under p in t.